Projet - Réorganisation d'un réseau de fibres optiques.

de Mubiligi Bettina et Gourdette Beeverly (groupe 4).

Ce projet a pour objectif la mise en place la gestion d'un réseau de fibres optiques d'un point de vue informatique et performance. On s'intéresse à la visualisation d'un réseau grâce à différents outils (texte, affichage SVG...) mais également aux différentes manières possibles pour implémenter l'environnement de travail et leur efficacité (Arbre, Listes chaînées, Table de hachage...).

Sommaire*:*

Partie 1 - Lecture, affichage et stockage des Données

---

1. Chaines.h : Lien entre réalité et modélisation

Le header Chaines.h permet de faire le lien entre l'environnement réel du réseau de fibres optiques et les différentes modélisations informatiques. Nous utilisons comme base le fichier *"00014_burma.cha"* qui correspond à une représentation simplifiée du réseau de Birmanie. Nous avons décidé d'implémenter plusieurs outils qui nous permettent de mettre en place notre représentation à partir d'un fichier cha.

Tout d'abord, voici les explications des structures composant Chaines.h et leur rapport avec la réalité.

- **cellPoint** : Liste chaînée composée de coordonnées de type double x et y . Elle correspond à une liste de points du plan par exemple la liste des coordonnées des clients ou des concentrateurs sur la carte d'une ville.
- **cellChaine**: Liste chaînée comprenant un numéro unique qui lui est attribué et une liste de points. Correspond au câble d'un réseau
- **Chaînes** : Structure possédant les caractéristiques du réseau : son gamma (nombre maximal de fibres utilisables par un câble), le nombre de câbles et la liste des câbles.

Ces structures nous permettent de manipuler les données grâce aux fonctions implémentées. Chacune de ces fonctions jouent un rôle dans les grandes catégories suivantes :

1. Fonction de manipulations des fichiers

permettent de passer du fichier à la structure et inversement, également représentation sous forme d'image SVG.

lecturesChaines
ecrireChaines
affichageChainesSVG

2. Fonction de création de structures
   permettent d'allouer la mémoire nécessaires pour les structures citées précédemment

   creerChaines
   creeCellChaine
   insertCellPoint

3. Fonction de libération des Structures
   permettent d'assurer aucune fuite mémoire après création et utilisation des structures

   libererChaines
   libererCellChaines

4. Fonction de manipulation des structures
   permettent d'avoir des informations sur les structures, outils.

   affichage
   longueurChaine
   longueurTotale
   comptePointsTotal
   generationAleatoire

Ces fonctions nous permettent et ces structures doivent être solides car elles sont les bases pour l'objectif majeur du projet : la reconstitution du réseau. Nous avons donc implémentés trois manières de passer des Chaines à une structure de réseau viable : La liste chainée, la table de Hachage et l'Arbre Quaternaire.

Partie 2 - Reconstitution du Réseau

---

1. Reseau.h : La structure au cœur du projet.

Le fichier header Reseau.h contient les structures indispensables pour effectuer la reconstitution d'un réseau réel:

- **cellNoeud**: Liste chaînée contenant un pointeur vers un Noeud et un autre vers un autre cellNoeud. Peut correspondre à tous les nœuds du réseau mais également à la liste des voisins d'un nœud. D'un point de vue réel, le cellNoeud correspond à un à une liste de clients ou de concentrateurs.
- **Noeud**: Structure comprenant un numéro unique qui l'identifie, des coordonnées et une liste de voisins sous la forme de cellNoeud. Le Noeud représente à un client ou un concentrateur.
- **cellCommodite**: Liste chaînée ayant comme données deux Noeuds extrA et extrB correspondant aux deux extrémités d'un câble et un pointeur vers un autre cellCommodite
- **Reseau**: Structure contenant toutes les informations liés à un réseau, le nombre de nœuds, le gamma, la liste des noeuds du réseau (= liste des clients et concentrateurs) et enfin la liste des extrémités de chaque câble.

Le fichier Reseau.h est accompagné du fichier Reseau.c contenant les fonctions suivantes :

1. Fonction de création des structures:

   creerNoeud
   creeCellNoeud
   creerReseau
   creerCellCommodites

2. Fonction donnant des informations sur le Reseau

   printCommodite
   afficheReseau
   nbCommodites
   nbLiaisons
   compteVoisins

3. Outils de manipulation des fichiers avec Reseau

   ecrireReseau
   afficheReseauSVG

4. Fonction de libération de la mémoire

liberercellNoeud
libererNoeud
libererReseau.

Reseau.c a également une multitude de fonctions destinées à implémenter une méthode utilisant une reconstitution du réseau à l'aide d'une liste chainée afin d'aider les opérateurs à organiser leur réseau de fibre.

2. Reseau.h : Restitution du Réseau par Liste Chaînée

5. Fonction de mise en relation des structures Chaines et Reseau à l'aide d'une Liste Chaînée

   rechercheCreeNoeudListe
   reconstitueReseauListe
   InVoisins
   fusionVoisins
   addCommodites

Nous avons pris le parti d'ajouter des fonctions intermédiaires de gestion de de la liste cellNoeud des voisins (InVoisins, fusionVoisins). Afin d'alléger les codes et améliorer la lisibilité de la fonction reconstitueReseauListe. Elle a pour rôle majeur de reconstituer le réseau à partir d'une Chaîne C à travers une liste chaînée.

3. TableHachage.h : Restitution du Réseau par une Table de Hachage.

Une autre méthode d'implémentation est présente dans le fichier TableHachage.h et TableHachage.c. Une table de hachage permet de retrouver un élément dans un tableau grâce à une clé. Il est également nécessaire de gérer les collisions. C'est pourquoi dans notre implémentation, la table de hachage est un tableau de liste. Cette méthode nous permet de savoir d'avance si un nœud a déjà été mis dans le réseau ou non. Pour mieux comprendre, la structure que nous avons créée est la suivante :

- **TableHachage**: Structure contenant la taille du tableau ainsi qu'un tableau de pointeurs vers une liste de nœuds.

Des explications plus détaillée sur le tableau T est représenté sur le schéma ci dessous:

/* Insérer Schéma */

Nous avons besoin de fonctions pour mettre en place notre structure :

1. Outils pour créer la TableHachage

   key
   hachage
   creerTableHachage

2. Fonction de libération de la TableHachage

   libererTableHachage

3. Mise en relation de la structure Chaines et Réseau à travers une Table de Hachage

   reconstitueReseauHachage
   rechercheCreeNoeudHachage

Cette implémentation permet de couper le temps de recherche car grâce au tableau, nous avons déjà un indice sur la position du Noeud et s'il est dans le Réseau ou non grâce au système de clé et de tableaux de pointeurs. Cependant, il y a quand même une collision inévitable donc une recherche dans la liste de Noeud correspondant à la case du tableau nécessaire. Nous nous demandons maintenant si nous pouvons trouver une structure qui diminuerait encore plus le temps de calcul et qui ne serait pas soumis au collisions en gardant un aspect unique. Nous pensons donc à une implémentation en arbre : Les arbres quaternaires.

4. ArbreQuat.h : Restitution du Reseau par un ArbreQuat

Finalement, nous terminons notre étude sur les manières d'implémenter le réseau par un Arbre Quaternaire, c'est-à-dire un arbre qui possède 4 fils. Les fils sont représentés à partir des coordonnées du centre de la cellule : ils sont alors au nord-est, sud-est, nord-ouest ou sud-ouest à partir du point de vue du centre. On traverse donc l'arbre en comparant les coordonnées x et y jusqu'à trouver le Noeud ou bien directement la place qui lui est due. Cette structure est basée sur l'orientation et le gain de temps en insertion car l'action de recherche lui est complémentaire. Quelques calculs sont nécessaires en utilisant l'arbre parent mais aucune ne nécessite d'itérations lourdes dans une liste. Voici les fonctions que nous avons implémentées pour supporter la structure ArbreQuat ci dessous.

- **ArbreQuat** : Structure composée de coordonnées d'un centre d'une cellule, les dimensions de celle ci (coteX et coteY), un Noeud qu'elle contient et de ses fils.

1. Outils pour créer un ArbreQuat

   chaineCoordMinMax
   creerArbreQuat

2. Fonction de libération d'un ArbreQuat

   libererArbreQuat

3. Outils de mise en relation Chaine et ArbreQuat

   updateCentre
   updateSens
   insererNoeudArbre
   rechercheCreerNoeudArbre

4. Outils de mise en relation Chaine, Reseau et ArbreQuat

   reconstitueReseauArbre

En conclusion, nous avons vu trois implémentations à leur avantage : la liste Chainee, la table de Hachage et l'arbre Quaternaire. Avant d'observer les performances en temps d'exécution, nous allons implémenter un système de choix et d'écriture pour l'opérateur du réseau de fibre optique afin de lui laisser le choix.

Partie 3 : Ecriture d'un Réseau.

Nous voulons à présent construire des méthodes pour manipuler et afficher le Réseau que nous avons construit à travers divers moyens. Une nouvelle fois, le fichier "00014_burma.cha" est notre

base. nous avons notamment besoin des fichiers nbCommodites, ecrireReseau et affichageSVG du fichier Reseau.c

La forme du fichier est la suivante :
Les quatre premières lignes donnent les informations sur le Reseau, nombre de Noeud, nombre de Liaisons, nombre de Commodités et gamma.
Ces lignes sont suivies par tous les nœuds du réseau représentés par leur coordonnées précédées par un v.
Les lignes commençant par un l contiennent un câble donné par les numéros des deux extrémités
Enfin nous avons la lettre k suivie par toutes les commodités du réseau.
Nous allons tester nos 3 implémentations dans le fichier ReconstitueReseau.c

1. ReconstitueReseau.c : Visualisation d'un réseau dans un fichier.

Le programme utilise la ligne de commande pour prendre un fichier .cha en paramètre et un nombre entier indiquant quelle méthode utilisée (1 - Liste chaînée, 2 - table de hachage, 3 - Arbre). Dans chacune des méthodes, on vérifie le bon fonctionnement de chaque méthode. les fichiers burmacpyARBRE.cha, burmacpyLISTE.cha, burmaHACHAGE.cha écrivent dans le fichier le réseau créé grâce à leur méthode respective, témoignant du bon déroulement de la transition Chaine à Réseau avec chaque méthode.

2. GrapheMain.c : Visualisation du graphe.

Ce programme commence par créer un réseau à l'aide du fichier 00014_burma.cha fourni qui contient des nœuds. On crée ensuite le graphe associé au réseau et on l'affiche. Ce programme peut également servir à tester les autres fonctions de gestion des graphes, comme reorganiseReseau, ou encore nbAretes.

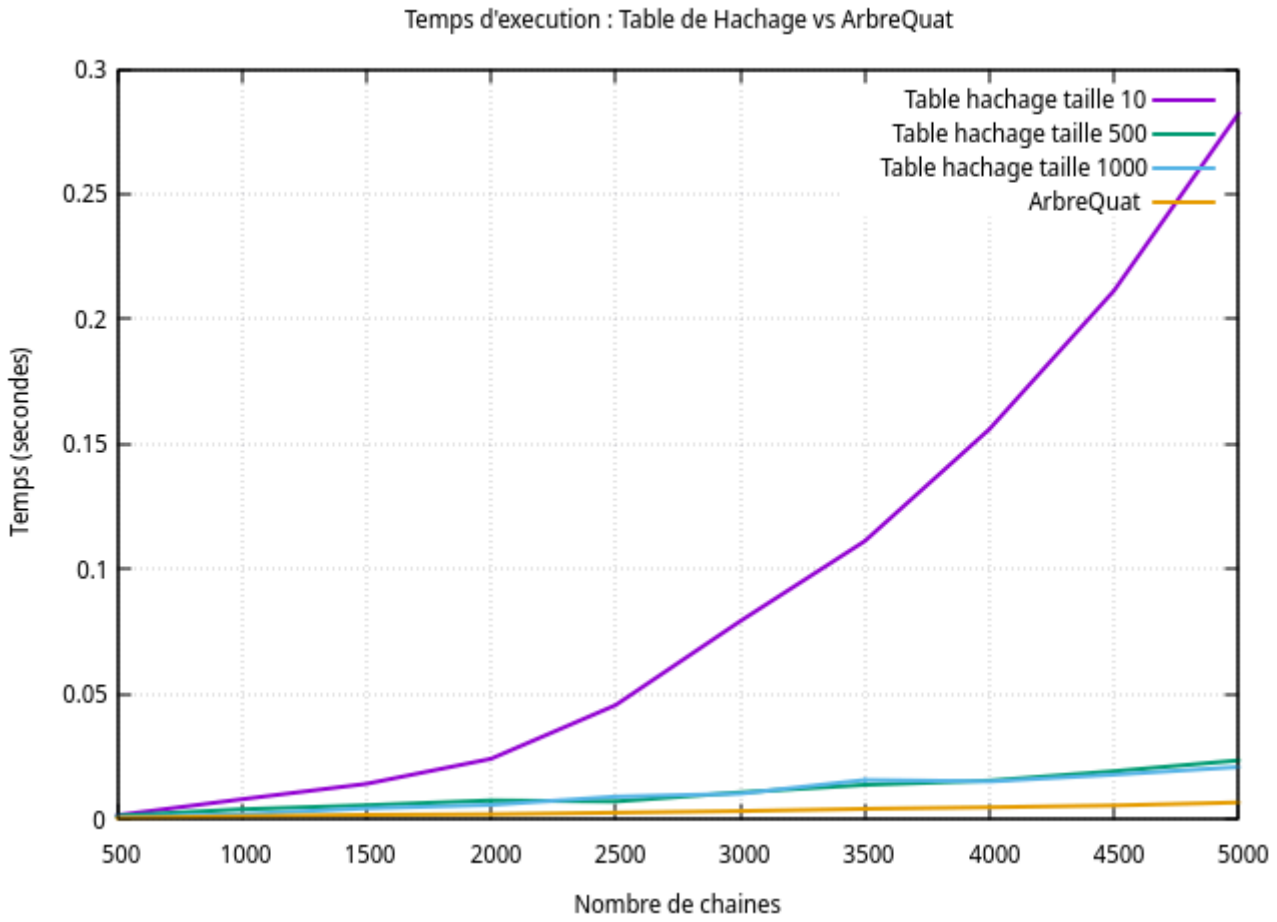
Partie 4 : Comparaison des trois structures en terme de performance

---

1. **<u>ReconstitueReseauComp :</u>** Représentation visuelle de l'efficacité des structures.

On définit le nombre d'appels et de pas à 5000 et 500 respectivement. On crée deux fichiers qui vont contenir nos données : un fichier pour les données Liste et un autre pour les données des tables de hachage et des arbres. On observe le comportement de trois tables de hachage pour plus de lisibilité : taille 10, 500 et 1500. On a donc les graphes suivant qu'on obtient grâce au script gnuplot scriptHachageArbre.gp et scriptListe.gp :

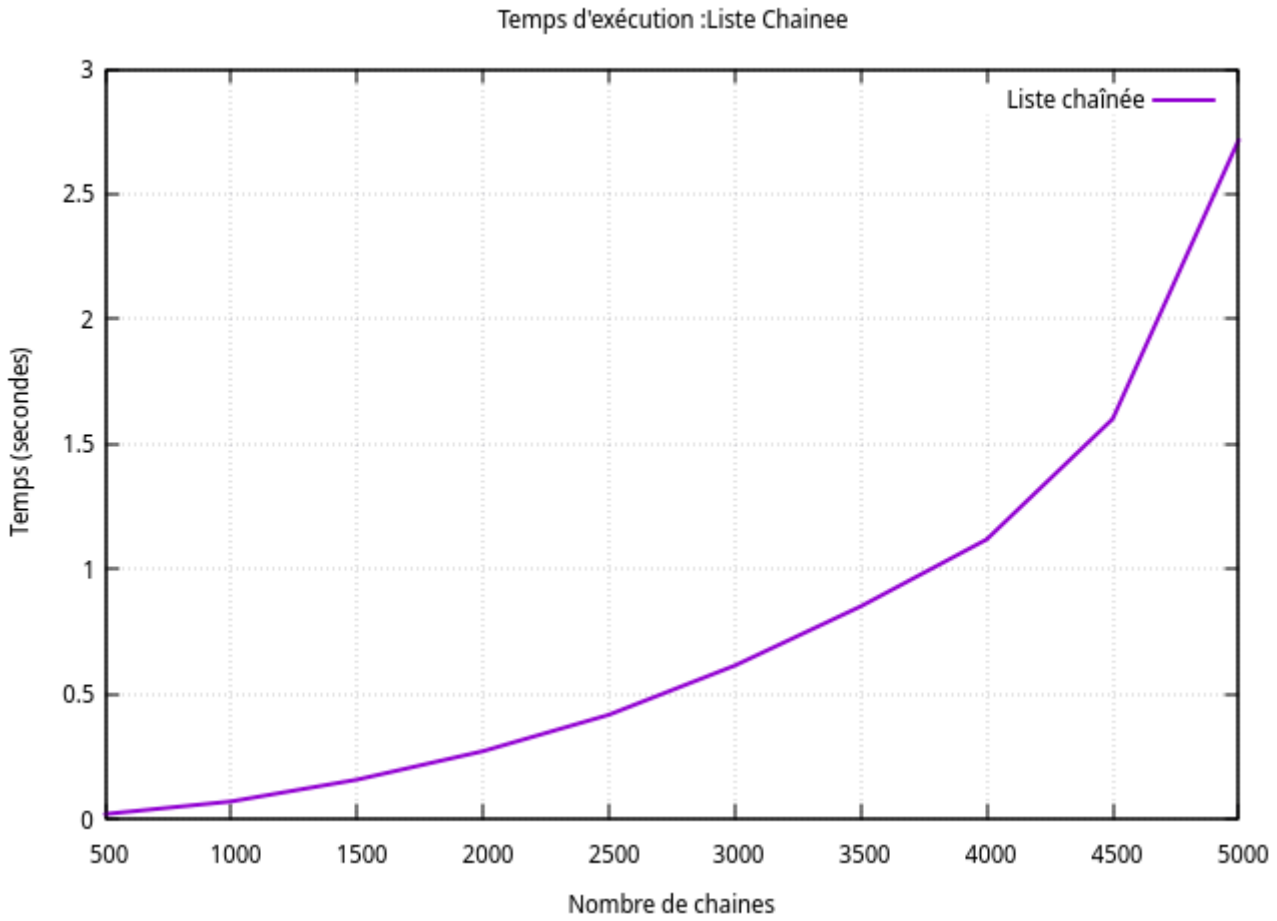
2. <u>Temps d'exécution :</u> Table de Hachage vs ArbreQuat



On remarque que plus la table de hachage est petite , plus le temps d'exécution est lent, choisir une taille 10 serait une perte de temps notable lorsqu'on regarde les performances des tables de taille

1000 qui est en deçà de 500. Toutefois, celles-ci ne sont pas comparables aux performances de l'ArbreQuat qui a un temps d'exécution quasiment instantané. Il est donc évident que si l'opérateur cherche une implémentation très rapide, il devrait prendre l'ArbreQuat. Cependant, il est également important de prendre en compte l'ergonomie et l'accessibilité du point de vue du client. ll peut être pertinent de prendre une table de hachage pour laisser au client le choix de la taille pour la personnalisation ou bien une compréhension plus facile de l'implémentation. En effet, il est important de prendre en compte les différents aspects, l'arbreQuat est très efficace mais il peut être difficile pour l'opérateur de comprendre les mécanismes de l'arbre, ce qui pourrait le ralentir lors des maintenances des fibres optiques par exemple. Il reste encore à l'ArbreQuat un concurrent : la liste chaînée que nous observons au prochain graphe.

3. Temps d'exécution : Liste Chainée



Les performances de la liste chaînée sont très en arrière des deux implémentations atteignant un temps d'exécution de 3 secondes qui est bien au-dessus des temps d'exécutions de moins d'une seconde de la table de hachage et de l'arbreQuat. Le problème est que la liste chaînée n'a pas de stratégie pour un recherche du nœud aussi efficace que la table de hachage ou l'arbreQuat ce qui

explique son temps aussi lent. Cette méthode est donc écartée pour l'opérateur car elle ne présente pas d'avantages pour le client : manque de personnalisation et d'efficacité en termes de temps d'exécution.

Partie 5 : Stockage du Réseau dans un graphe

1. **Graphe.h :** Une représentation sous forme de graphe

Le fichier graphe.h contient la structure de graphe :

- **Arete** : une arête est une chaîne entre deux sommets du graphe, c'est l'équivalent d'un câble du réseau. La structure prend la forme de deux entiers u et v représentant les numéros des sommets qui forment l'arête.
- **Cellule_arete** : cette structure stocke une liste d'arêtes, avec une cellule contenant une arête, et l'arête suivante dans la liste.
- Sommet : la structure sommet du graphe est l'équivalent d'un nœud dans le réseau. Un sommet de graphe a un numéro, deux flottants x et y qui représentent ses coordonnées, ainsi qu'une liste de Cellule_arete qui contient la liste des arêtes qu'il forme avec ses voisins.
- **Commod** : la structure commodité d'un graphe permet de stocker les commodités, comme avec le réseau, sous forme de deux entiers qui représentent les numéros des sommets qui forment la commodité.
- **Graphe** : rassemble toutes les structures. Un graphe a 3 valeurs entières : un nombre de sommets, gamma et le nombre de commodités, un tableau de pointeurs qui pointent chacun sur une structure sommet, soit un sommet par case, et un tableau de commodités qui stocke toutes les commodités du réseau.

1. Fonctions de création de structures
   - **creerGraphe**
   - **creerGraphe2**
   - **creerCelluleArete**
   - **creerSommet**
   - **creerArete**
2. Fonctions de gestion des structures :
   - **Liste_chaine** : retourne une liste d'entiers correspondant à la plus petite chaîne entre des sommets u et v du graphe.
   - **reorganiseReseau** : retourne vrai si pour toute arête du graphe, le nombre de chaînes qui passe par cette arête est inférieur à gamma, et faux sinon.

- **rechercheArete** : recherche une arête entre les sommets u et v d'un graphe dans la matrice stockant les arêtes de ce graphe.
- **afficherGraphe :** affiche la structure graphe complète.
- **nbAretes** : retournant le plus petit nombre d'arêtes d'une chaîne entre deux sommets u et v d'un graphe.

3. Fonctions de libération des structures
   - **libererGraphe**
   - **libererCelluleArete**
   - **libererSommet**

Partie 6 : Réponses aux questions

Q4.2 - Pour l'implémentation d'une table de Hachage utilisant comme clé l'utilisation de la fonction keyF sur les coordonnées x et y d'un point P est approprié. En effet, si nous prenons par exemple des points x et y entre 1 et 10 aléatoirement, nous remarquons que les clés sont toutes différentes. En conséquence les risques de collisions sont faibles ce qui signifie un gain de performance lors de la recherche d'un point (Moins de recherches dans les sous-tableaux de T). Les clés obtenues à partir des points provenant de la base de données burma_cha ont un comportement similaire. Ainsi nous prenons quelques points avec les coordonnées arrondis à l'entier inférieur, les clés sont toutes différentes.

Q6.1 - On observe que les temps de reconstitution du réseau avec arbre semblent être plus courts en moyenne qu'avec la liste. Par hachage, on remarque qu'en fonction de la taille de la table de hachage, le temps varie : plus la table est grande, moins la reconstitution du réseau prend de temps, et plus son temps se rapproche de celui de l'arbre quaternaire. Cela pourrait s'expliquer par un nombre de collisions faible avec les tables plus grandes.

Q6.4 - En étudiant les graphiques, on observe que plus le nombre de points total des chaînes augmente, plus le temps pour reconstituer le réseau augmente en conséquence. Il semblerait également que nos observations passées soient confirmées : la reconstitution d'un réseau avec un arbre est en moyenne plus rapide et efficace que celle avec une table de hachage, qui est influencée par la taille de la table. En effet, en observant le 2e graphique, on voit que plus la table de hachage est grande, moins le temps CPU pour la reconstitution d'un réseau sera élevé. Il faudrait que la taille de la table de Hachage soit la plus grande possible pour avoir un temps de reconstitution optimal. De plus, en s'appuyant sur le graphique de la liste chaînée, on observe que le temps de reconstitution de la liste est très similaire à celui de la table de hachage de taille minimale (ici 10), c'est donc la méthode la moins optimale. La méthode la plus efficace est donc l'arbre quaternaire.